

# Laravel Tesztelés

# Smart24

## Tartalom

<b>1. LARAVEL KERETRENDSZER.....</b>	<b>2</b>
<b>2. LARAVEL TESZTEK.....</b>	<b>2</b>
<b>3. SMART24.....</b>	<b>3</b>
<b>4. LARAGON.....</b>	<b>4</b>
<b>5. UNIT TESTS.....</b>	<b>4</b>
5.1 Modell tests .....	5
5.2 Controller tests.....	5
<b>6 FEATURE TESTS.....</b>	<b>8</b>
<b>7 TESZT EREDMÉNYEK .....</b>	<b>9</b>
<b>8 IRODALOMJEGYZÉK .....</b>	<b>10</b>

Készítette:

**Lukács Szabolcs, Aut. IV.**

## 1. Laravel keretrendszer

A Laravel egy nyílt forráskódú, ingyenes, modern webalkalmazás-keretrendszer, melyet Taylor Otwell fejlesztett ki. A PHP nyelven íródott keretrendszer sok alapvető funkciót és szolgáltatást tartalmaz, amelyek lehetővé teszik a gyors és hatékony webalkalmazások fejlesztését. A Laravel magas szintű biztonságot és támogatást kínál, így könnyű és hatékony fejlesztést tesz lehetővé. A keretrendszer nagyon népszerű, és használható kis- és nagy projektekhez egyaránt.

## 2. Laravel tesztek

A Laravel keretrendszer nagyon erős tesztelési támogatást nyújt, amely lehetővé teszi a webalkalmazások hatékony tesztelését és hibák kiszűrését. A Laravel-ben használható tesztek PHPUnit-ra épülnek, ami egy PHP alapú tesztelő keretrendszer.

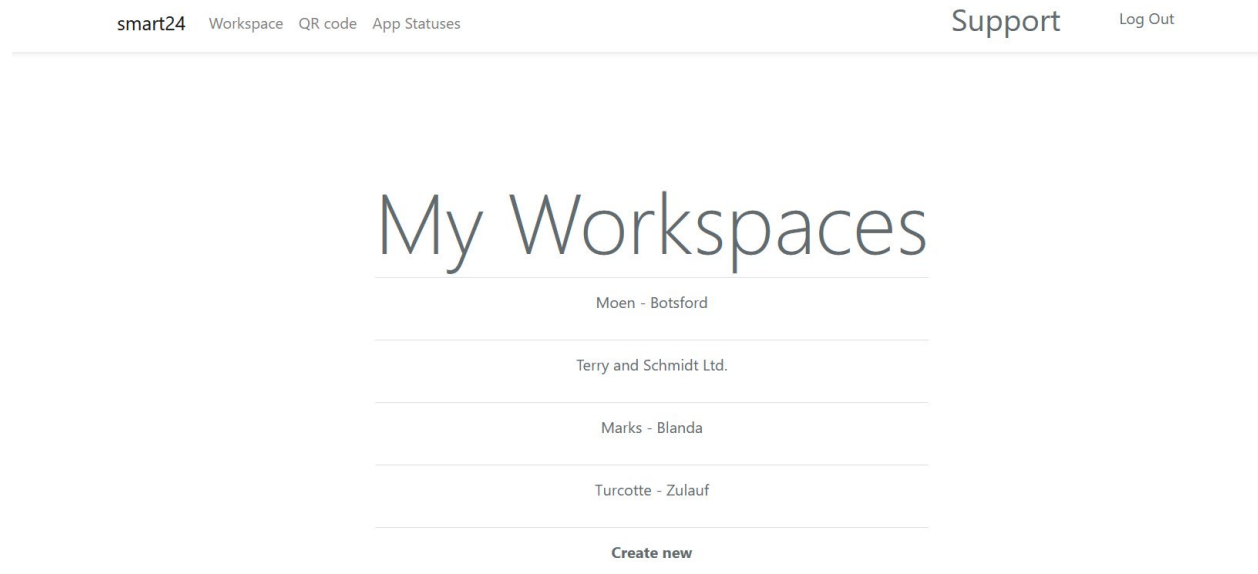
A Laravel tesztelési funkciója lehetővé teszi a különböző tesztesetek definiálását, amelyek a webalkalmazások egyes részeinek tesztelésére szolgálnak. Például a tesztelhető részek lehetnek az adatbázis műveletek, az adatok validálása, a felhasználói felület működése, a HTTP kérések és válaszok tesztelése, stb.

A Laravel tesztelési funkciója segítségével a fejlesztők automatikusan tudnak tesztelni, így időt és pénzt spórolva meg, valamint biztonságot és megbízhatóságot biztosítanak a webalkalmazások számára. A Laravel tesztelési funkciója számos előre definiált tesztsztyalt és tesztelési lehetőséget biztosít, amelyek lehetővé teszik a fejlesztők számára, hogy hatékonyan és könnyen tesztelhessék a kódjukat.

A tesztek futtatása egyszerű a Laravel-ben. A tesztelési funkcióhoz először létre kell hozni egy tesztesetet, majd definiálni a teszteset műveleteit, majd futtatni a tesztek a PHPUnit segítségével.

### 3. Smart24

A Smart24 egy webalkalmazás, amely egy kisvállalkozások számára kínál hasznos funkciókat. A projekt egy szakmai gyakorlaton kezdődött el, ahol egy webfejlesztő cég keretein belül folyt a fejlesztés. Az oldal célja az, hogy egyszerű és hatékony felületet biztosítson a felhasználók számára, akik sok hasznos funkciót használhatnak.



**ábra 3.1.** Példaként létrehozott munkaterek

Az oldal jelenlegi funkciói közé tartozik a felhasználói regisztráció és bejelentkezés, amely lehetővé teszi a felhasználók számára, hogy hozzáférjenek a szolgáltatásokhoz. A felhasználók munkaterületeket hozhatnak létre, szerkeszthetik és törölhetik azokat, valamint alkalmazásállapotokat hozhatnak létre, szerkeszthetik és törölhetik azokat. Úgynevezett appstatusokat hozhatnak létre, ahol bizonyos funkciók állapotát figyelhetik, láthassák a módosítások dátumát. Az oldal további funkciója a QR-kód generálása, amely lehetővé teszi a felhasználók számára, hogy egyszerűen és gyorsan hozzáférjenek a weboldalukhoz.

# Terry and Schmidt Ltd.

Terry and Schmidt Ltd. 67570 Satterfield Parkway 586-587-8877 your.email+fakedata33064@gmail.com Eius non et et laudantium nostrum quam quasi.

## Members of workspace:

Support

Edit

Invitations to Workspace

Delete

### **ábra 3.2.** Egy munkatérhez tartozó adatok

Az oldal számos hasznos funkciót biztosít a felhasználók számára, amelyek könnyen kezelhetők és megfelelnek a kisvállalkozások igényeinek. Az oldal funkciói könnyen elérhetők a felhasználók számára, akik gyorsan és hatékonyan tudnak dolgozni az oldalon. A Smart24 egy nagyszerű lehetőség kisvállalkozások számára, akik hatékonyan szeretnék kezelni a munkafolyamataikat.

## 4. Laragon

A weboldalt a Laragon alkalmazás segítségével teszteltem a saját számítógépeimen. A Laragon egy könnyen használható fejlesztőkörnyezet, amely lehetővé teszi, hogy könnyedén futtassunk egy lokális szervert a saját gépünkön.

## 5. Unit tests

A unit tesztjeim két fő részre vannak osztva: Modells, Controllers. A Modell-ben a modellek közti kapcsolatokat ellenőrzöm, például, hogy egy felhasználóhoz több munkatér is tartozhat. A Controllers-ben leginkább az oldal funkciói tesztelem. Például a QR generálás funkciót.

## 5.1 Modell tests

4 fő osztályom van:

- AppStatusTest
- UserTest
- WorkspaceInvitationTest
- WorkspaceTest

Ezekben rendszerint létrehozok egy-egy példányt az adatbázisban, majd létrehozom köztük a kapcsolatot. Az assertekben ellenőrzöm, hogy megfelelő-e a köztük lévő kapcsolat.

```
public function test_work_space_belongs_to_many_user()
{
    DB::table( 'user_work_space' )
        ->insert([
            'user_id' => User::factory()->create()->getKey(),
            'work_space_id' => $this->workspace->id,
        ]);
    $this->assertInstanceOf( expected: User::class, $this->workspace->users->first());
}

public function test_work_space_has_many_invitations()
{
    WorkspaceInvitation::factory()
        ->for($this->workspace)
        ->create();
    $this->assertInstanceOf( expected: WorkspaceInvitation::class, $this->workspace->workspaceInvitation->first());
}
```

*ábra 5.1.* Két példa teszt, ami a modellek közti kapcsolatokat ellenőrzi

## 5.2 Controller tests

4 fő osztályom van:

- AppStatusControllerTest
- QRCodeControllerTest
- WorkspaceInvitationControllerTest
- WorkspaceControllerTest

Mindegyik modellemnek az általános funkciói:

- index()
- show()
- update()
- edit()
- destroy()

```
public function test_appstatus_can_show_an_appstatus(): void
{
    $user = User::factory()->create();
    $this->workspace = Workspace::factory()->create();
    // Add this workspace to this user
    DB::table('user_work_space')
        ->insert([
            'user_id' => $user->id,
            'work_space_id' => $this->workspace->id,
        ]);
    $appstatus = AppStatus::factory()
        ->for($this->workspace)
        ->create([
            'name' => 'Test appstatus name',
            'id' => 10
        ]);
    $response = $this->actingAs($user)
        ->get('uri: 'appstatus/10');
    $response->assertSee('value: 'Test appstatus name');
}
```

ábra 5.2. AppstatusControllerTest

Az (ábra 5.2) ábrán látható tesztben létrehozok egy felhasználót, ennek a felhasználónak egy munkateret, és egy appstatust, ami a munkatérhez tartozik. Az actingAs(user) a login műveletet végzi el automatikusan, hogy a tesztben a felhasználó már bejelentkezve érje el az adott oldalt. Az assertSee-vel ellenőrzöm, hogy az oldalon megjelent-e az azelőtt létrehozott egyedi appstatus.

```
public function test_function_generate_empty_url(): void
{
    $emptyt_url = '';

    $response = $this->post( uri: 'qrcode', ['url' => $emptyt_url]);
    $response->assertStatus( status: 200);
    $response->assertSee( value: 'Generate a QR code');
}
```

**ábra 5.3.** Empty Qrcode Generation Test

Az oldal akkor működik helyesen, ha a felhasználó nem adott, meg semmit a QRcode generálás mezőben, akkor maradjon ugyan azon az oldalon. A ‘Generate a QR code’ felirat csak a kezdő oldalon látható, tehát ha a teszt továbbra is lássa a feliratot az oldalon, akkor helyes a működés.

```
public function test_function_generate_normal_url(): void
{
    $url = 'https://ms.sapientia.ro/hu';

    $response = $this->post( uri: 'qrcode', ['url' => $url]);
    $response->assertStatus( status: 200);
    $response->assertSee( value: 'Your QR Code');
}
```

**ábra 5.4.** Qrcode Generation Test

Az (ábra 5.4) ábrán látható tesztben egy valós url cím generálást tesztel a kód. A ‘Your QR Code’ felirat csak a kigenerált QR kód mellett látható. Amennyiben a 200-as státuszt kapunk vissza és látható a felirat, akkor a teszt sikeresnek bizonyult.

## 6 Feature tests

A feature tesztelés a Laravel keretrendszerben egy olyan tesztelési folyamat, amely során a különböző funkciókat, illetve jellemzőket teszteljük, amelyeket a felhasználók valósítanak meg. A feature tesztek célja, hogy biztosítsák a Laravel alkalmazás helyes működését és végfelhasználóbarát legyen.

Itt két fő funkciót ellenőriztem:

- Login
- Register

```
public function test_login_screen_can_be_rendered(): void
{
    $response = $this->get( uri: '/login');

    $response->assertStatus( status: 200);
}

public function test_users_can_authenticate_using_the_login_screen(): void
{
    $user = User::factory()->create();

    $response = $this->post( uri: '/login', [
        'email' => $user->email,
        'password' => 'password',
    ]);

    $this->assertAuthenticated();
    $response->assertRedirect( uri: RouteServiceProvider::HOME);
}
```

*ábra 6.1.* Login funkció tesztelése

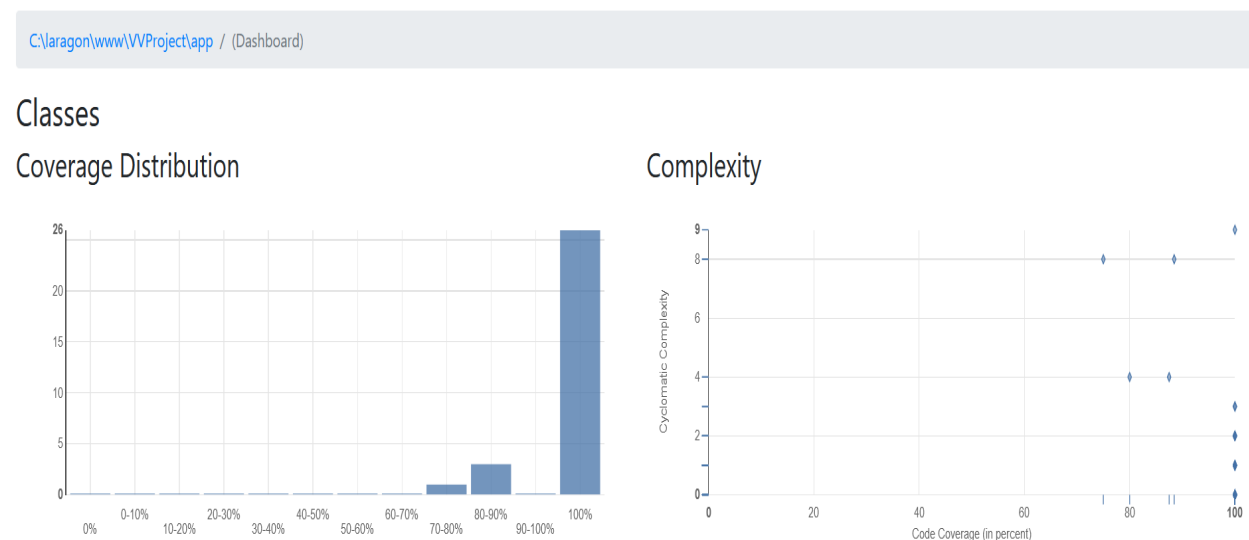
A (ábra 6.1)-n látható **test\_login\_screen\_can\_be\_rendered()** tesztben azt ellenőrzöm, hogy a login oldal betöltődik-e? A teszt elnavigál a '/login' oldalra. Ha 200-as választ kap, átment a teszten.



A `test_user_can_authenticate_using_the_login_sreen()` – ben létrehozok egy user-t, a korábban megírt factory segítségével. Ennek a usernek az email és jelszó kombinációjával megpróbálom bejelentkezni. Az `assertAuthenticated` ellenőrzi, hogy megfelelően bejelentkezett-e a user. Az `assertRedirect` ellenőrzi, hogy a sikeres bejelentkezés után a `RouteServiceProvider`-ben meghatározott helyre navigált-e az oldal.






















## 7 Teszt eredmények

A Laravel keretrendszer tesztelési funkciói mellé letölthető egy bővítmény, `xDebug`. Ennek a segítségével egy reportot, jelentést lehet generáltatni automatikusan a lefuttatott tesztekéről. Ezekről különböző statisztikákat készít, minden fájlban mutatja részletesen a kódlefedettségét.



**ábra 7.1.** Teszt lefedettség eloszlása

A (ábra 7.1) ábrán látható, hogy a legtöbb teszt 100%-s lefedettséget nyújt, lefedi az adott kódrész 100%-át. Összesen 35 teszt van, az összes sikeresen lefut.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		92.97%	119 / 128		90.00%	36 / 40		80.00%	16 / 20
■ Console		100.00%	2 / 2		100.00%	1 / 1		100.00%	1 / 1
■ Exceptions		100.00%	1 / 1		100.00%	1 / 1		100.00%	1 / 1
■ Http		92.59%	100 / 108		87.50%	21 / 24		62.50%	5 / 8
■ Models		100.00%	8 / 8		100.00%	8 / 8		100.00%	5 / 5
■ Policies		75.00%	3 / 4		50.00%	1 / 2		0.00%	0 / 1
■ Providers		100.00%	5 / 5		100.00%	4 / 4		100.00%	4 / 4

### Legend

Low: 0% to 50%    Medium: 50% to 90%    High: 90% to 100%

Generated by [php-code-coverage 9.2.15](#) using [PHP 8.1.3](#) and [PHPUnit 9.5.21](#) at Sun Mar 26 12:19:16 UTC 2023.

## ábra 7.2. Kód lefedettség részletesen, egységenként

Az (ábra 7.2)-n látható részletesebben az összes mappa kód lefedettsége. Az eredeti program tartalmaz egy Admin mappát is, ahol az adminisztrátori joggal rendelkező felhasználókkal kapcsolatos funkciók találhatóak. Mivel ez még nem került implementálásra, nem volt konkrét funkciója, ezért azt kivettem a code coverage-ből. A tesztekéről további részletek találhatóak a repository<sup>[6]</sup> mappájában.

## 8 Irodalomjegyzék

- [1]. <https://laravel.com/docs/10.x/installation>
- [2]. <https://medium.com/@anowarhossain/code-coverage-report-in-laravel-and-make-100-coverage-of-your-code-ce27ccc738>
- [3]. <https://stackoverflow.com/questions/14379569/force-exclude-files-from-phpunit-code-coverage>
- [4]. <https://laravel.com/docs/10.x/testing>
- [5]. <https://laragon.org/>
- [6]. <https://github.com/szabilukacs/VVProject/tree/main/tests/Coverage/html>